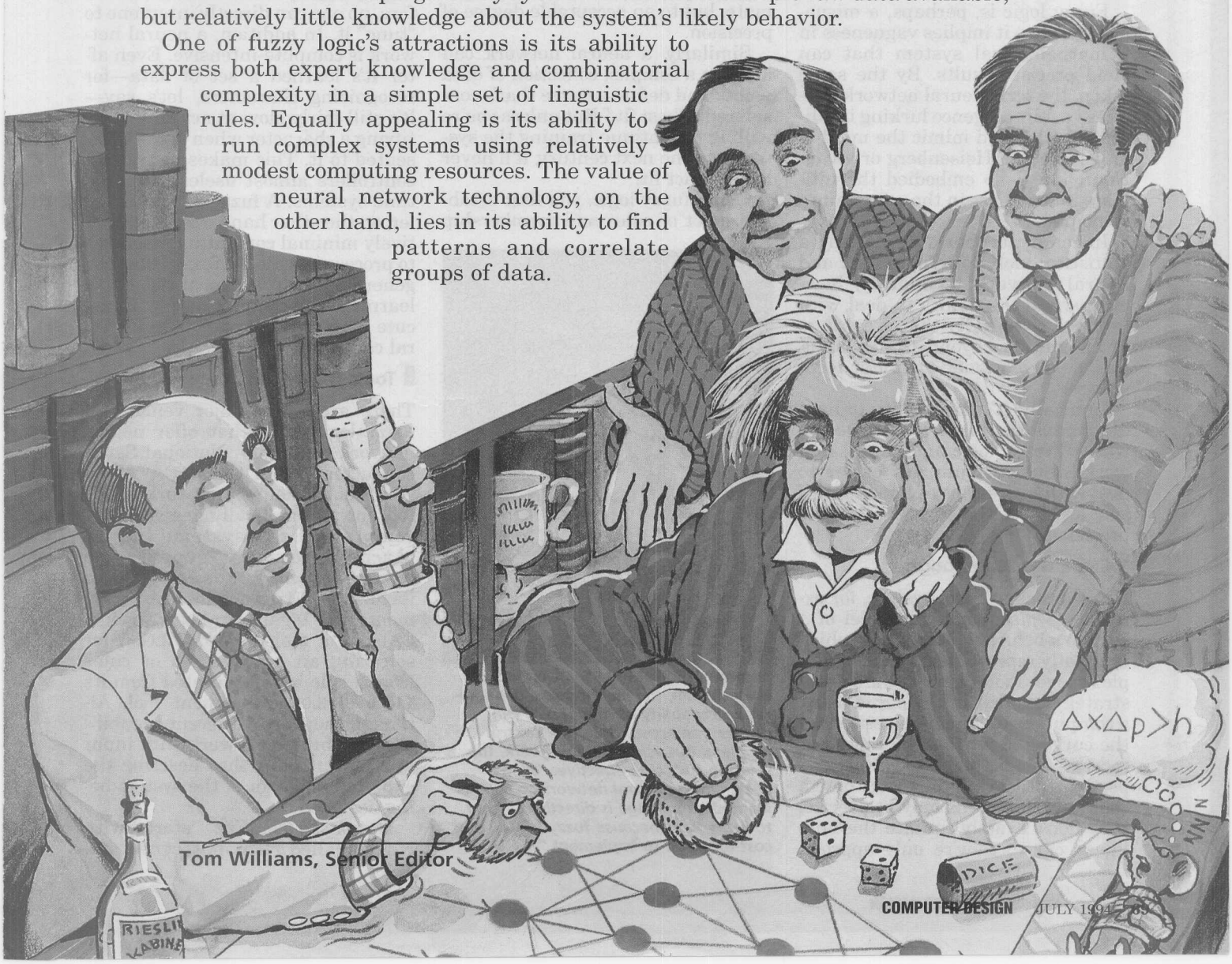
New tools make fuzzy/neural more than an academic amusement

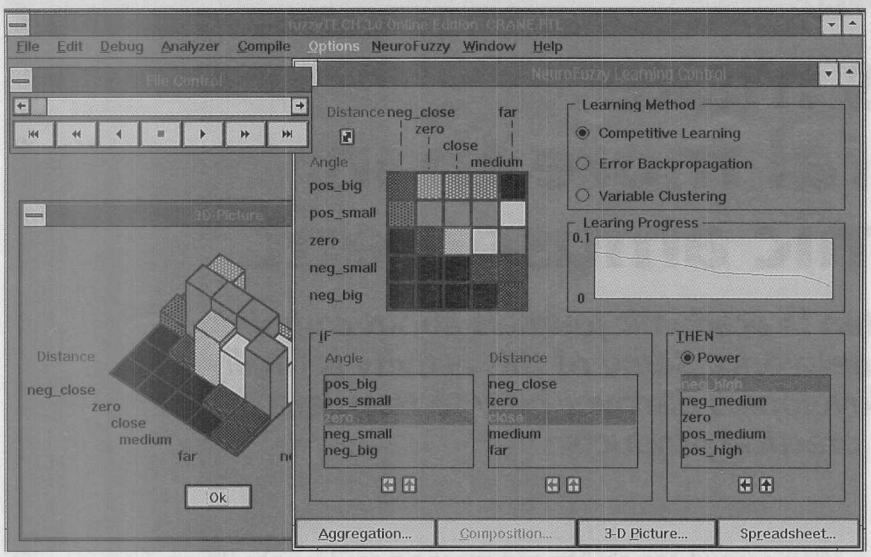
There's been a lot written in academic journals about combining fuzzy logic and neural networks. Some of this activity is finally coming to fruition as developers discover how to translate academic research into practical products.

Fuzzy logic tool vendors who offer neural capability are moving cautiously. There seems to be a fear that moving too quickly could confuse an engineering and software-development community that's still trying to get comfortable with some of the "radical" ideas behind fuzzy logic. Nevertheless, neural network learning can be a boon in developing a fuzzy system where there's ample raw data available, but relatively little knowledge about the system's likely behavior.

One of fuzzy logic's attractions is its ability to express both expert knowledge and combinatorial complexity in a simple set of linguistic rules. Equally appealing is its ability to run complex systems using relatively modest computing resources. The value of neural network technology, on the other hand, lies in its ability to find patterns and correlate groups of data.



Tom Williams, Senior Editor



The NeuroFuzzy module from Inform Software integrates with the company's fuzzyTECH fuzzy logic development tool. The NeuroFuzzy module can generate and optimize membership functions and rules out of sample or process data. You can watch the learning progress on the screen and intervene at any time, and manually edit the final results.

Fuzzy logic is, perhaps, a misnomer because it implies vagueness in a mathematical system that can yield precise results. By the same token, the term neural networks implies vast intelligence lurking in silicon, which could mimic the mind of an Einstein, a Heisenberg or a von-Neumann (who embodied the ultimate in fuzziness in the uncertainty principle).

In reality, a neural net is only a statistical tool. Both fuzzy logic and neural network technology, however, share the ability to deal with levels of complexity that are orders of magnitude beyond the capabilities of conventional computer technologies.

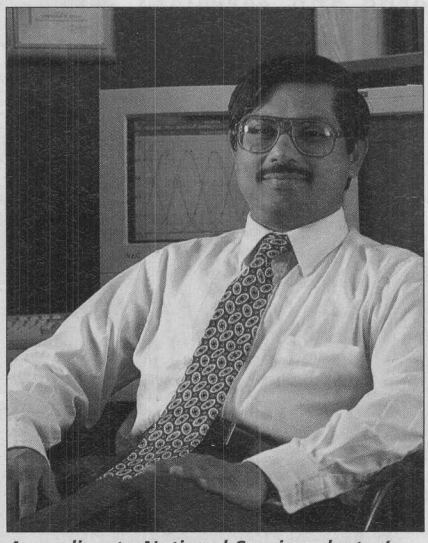
Ironically, it's this ability to handle complexity that makes people wary of fuzzy logic and neural networks. But in truth, it's easier to deal with high levels of complexity if you let go of the idea of absolute precision.

Close is good enough

In a classical control system, for example, a mathematical model of a system's behavior is created, which is usually represented by some complex curve. To implement the control strategy implicit in the curve using a computer, you linearize portions of the curve and write some computer code that implements the linearized portions of the curve. Even though the linear functions may be precise, they don't exactly imitate the nonlinear curve; they're only approximate, but to an acceptable degree of precision.

Similarly, a neural network can analyze a complex collection of data points and define a curve that's considered a "best fit." But unless you're willing to continue training the system into the next century, it'll never be an exact fit.

Using fuzzy logic, you first establish a set of rules and membership functions based on your best knowledge of how the system will behave. You then simulate and optimize until the system reaches acceptable behavior. All these approaches, even the ones you might think are precise mathematical models, rely on what Prof. Lotfi Zadeh, the inventor of fuzzy logic, calls "exploiting the tolerance for imprecision." In other words, a quantity can be correct even if it falls within a wide range of values.



According to National Semiconductor's Emdad Khan, the learning and generalization capability of neural nets can deliver a smart solution to a control problem. But as a neural network, the solution isn't cost-effective. "We've designed the neural network so that its learned knowledge is directly mapped to fuzzy logic because fuzzy is easy and cost-effective to implement."

When a control problem involves dynamic nonlinear systems, the two technologies working together can help manage the complexity and reduce design time given proper tool support. Two advantages are that neural nets can learn from raw data and their output can be used to generate a fuzzy rule base, which is a representation of a control strategy that you can look at and understand.

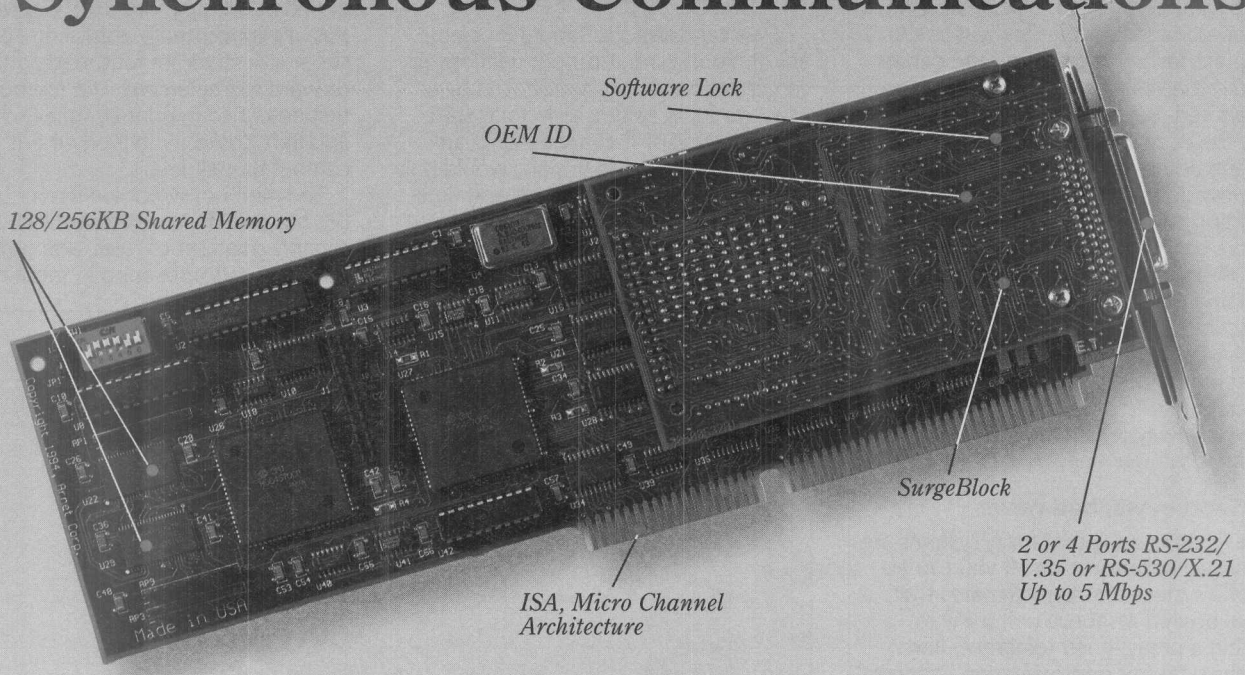
No one has insight into the "black box" of a neural network and, therefore, no one can directly intervene to "tune" it. In addition, a neural network is compute-intensive. Even after it's learned a set of data—for recognizing characters, let's say—it's still fairly slow at actually identifying a character when data is presented to it. This makes neural net controllers almost useless for real-time systems. A fuzzy logic controller, on the other hand, requires relatively minimal computing resources to process its rules. A fuzzy rule base generated from a neural net that's learned some complex data will execute many times faster than a neural controller.

Tools for neural/fuzzy

There are three major vendors of fuzzy logic tools that offer neural network capability: National Semiconductor (Santa Clara, CA) offers NeuFuz, a neural network-based tool for automatically generating a complete fuzzy system; Inform (Aachen, Germany) supplies the NeuroFuzzy module as an add-on to its fuzzyTECH development environment; and Togai InfraLogic (Irvine, CA) has a tool dubbed TILGen for selecting an optimal set of rules from a rule base generated from its TILShell 3.0 development tool. Although they have different capabilities, all three tools work with input and output data that describe the expected behavior of the system being designed.

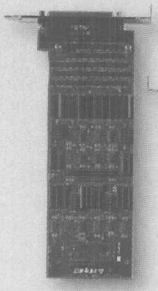
National's NeuFuz starts with what is called a training set of input-

SYNC/570i Expands the New Standard in Synchronous Communications



What Makes the SYNC/570i Better?
Arnet's SYNC/570 set the new standard in synchronous communications by offering the performance of an expensive coprocessor at a fraction of the price. Now the SYNC/570i expands on that standard by offering both two and four port configurations with multiple, user-configurable interfaces!

Unparalled Performance
Like the SYNC/570, the SYNC/570i utilizes the powerful Hitachi 64570 SCA chip. When coupled with the SYNC/570i's onboard memory, the 64570's full duplex DMA allows the board to support line speeds better than T1/E1. Even up to 5 Mbps for each line!



Your Choice of Interfaces
Available for both ISA and Micro Channel systems, the SYNC/570i can be configured for either two or four ports with RS-232/V.35 or RS-530/X.21 user selectable interfaces. Call today for more information on the growing list of software applications that now support Arnet's family of synchronous boards – ranging from the entry-level ArnetSync to the high performance SYNCom/3000 intelligent communications adapter.

Protect Your OEM Revenue
OEMs can customize the SYNC/570 and the SYNC/570i to include an OEM ID or machine-readable "fingerprint." Guarantee the protection of your software revenue by programming your application to this customized fingerprint.

Guaranteed Reliability
Both the SYNC/570 and the SYNC/570i are backed by our Rock Solid Lifetime Warranty. Plus, SurgeBlock onboard surge protection is available on all signal lines to guard against the leading cause of board failure – transient surges and spikes.

ARNET ROCK SOLID LIFETIME WARRANTY

Buy the Price/Performance Standard
Starting at only $595 for the SYNC/570 and $695 for the SYNC/570i, these products deliver a new price/performance standard unmatched by other PC-based synchronous communications controllers. Call **1-800-377-4401** to find out more.

ARNET
Rock Solid Connections from Digi International

618 Grassmere Park Drive Nashville, TN 37211
(615) 834-8000, (615) 834-5399, FAX

A look at adaptive fuzzy systems

Anyone who keeps contact with new developments in engineering has come across the term fuzzy logic. It's shown up at one time or another in most of the popular technical press and the debate about the value of fuzzy logic goes on unabated. But whatever that value might be, the issue will be resolved, ultimately, by the doers—not the talkers.

How do the doers achieve their goals? Starting with a set of system variables and a desired input-output relationship between them, they define fuzzy sets describing the variables and a set of rules describing the relationship. They succeed when the behavior of the resulting fuzzy system reproduces that relationship. Two questions lurk: In general, is it reasonable to always expect success, and if so, how do we manage it?

Good news, bad news

The good news is that fuzzy systems are universal approximators. If you can accept a system that approximates the ideal desired response uniformly to within a preassigned tolerance, Bart Kosko's "Fuzzy Approximation Theorem" says there are plenty of fuzzy systems that suffice. The only hitch is you have to find them on your own.

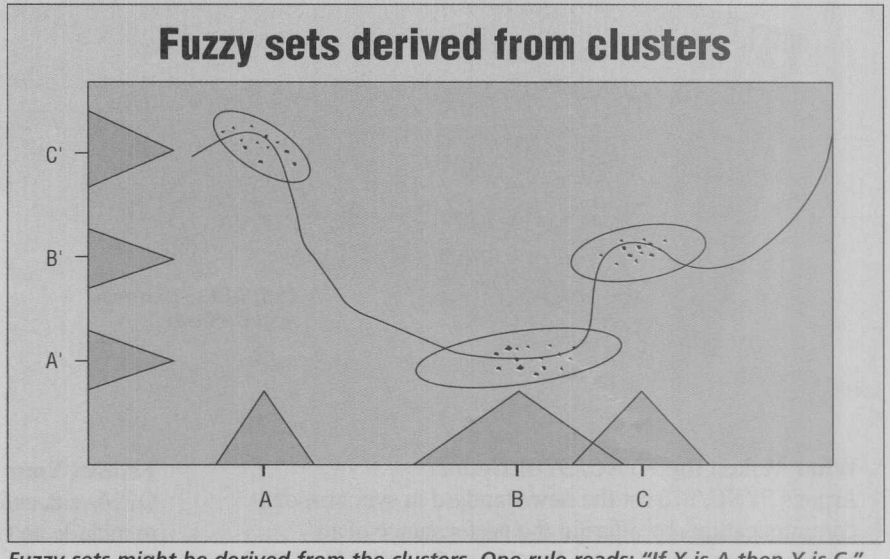
The bad news is that fuzzy systems are inherently approximate systems. There are very simple multivariate functions that can't be exactly realized by fuzzy systems, no matter how complicated. This means that a grand theory of representation by fuzzy systems is unattainable. Since perfect accuracy is often unreachable, even in principle, what we need is an automated way of realizing suitable approximate systems.

In principle, the algorithm of a fuzzy system is always the same. A fuzzy system derives its behavior from the rule and fuzzy set information it contains. In practice, this means that we need only determine parametric data for the fuzzy system—the rules and the fuzzy sets they involve. Sample data extracted from the desired underlying relationship is the basis for the determination.

Adaptive fuzzy systems are supposed to handle this optimization problem, which we call the Tuning Problem:

Given a tolerance level and samples of correct behavior, choose rules and fuzzy sets so that the resulting fuzzy system approximates correct behavior, to within tolerance, over the entire operating range of the system.

If we can solve the Tuning Problem, any stationary input-output relationship can be automatically reproduced by an adaptive fuzzy system as long as sufficient samples of that relationship are available. Handmade designs would only be needed for problems involving time-varying systems. Often, the engineering task would reduce to the generation of data for an adaptive fuzzy system to use.



Fuzzy sets might be derived from the clusters. One rule reads: "If X is A then Y is C," as is readily seen in the graph.

Currently, there isn't a general solution to the Tuning Problem. The problem is statistical, combinatorial and nonlinear—an analyst's nightmare. A typical problem can easily involve hundreds of numbers to adapt. Still, there are approaches that have been successful. Most of those appearing in the literature involve neural networks or genetic algorithms.

Generally speaking, these methods are instances of what is known as stochastic gradient descent. Consequently, they don't automatically find globally optimal solutions but only locally optimal ones. The difference is similar to the one between a molehill and a mountain range. At this stage, however, it's pretty much all we have.

Building an adaptive system

One particular way to build an adaptive fuzzy system has been proposed, but probably isn't ready for full automation because human intervention is usually required at one or more points. Nevertheless, it's conceptually appealing and the successive steps are reasonably straightforward to implement. The method was first described in print by Julie Dickerson and Bart Kosko, who worked out the computational details.

The method, which we'll refer to as DK, has two main objectives: Pick a reasonably good set of rules, and optimize the fuzzy sets appearing in these rules. Each objective uses a particular kind of neural network. To attain the first objective, DK uses competitive learning; for the second it uses supervised learning.

Competitive learning is an unsupervised distributed algorithm that estimates the underlying probability density of the data it sees. It occurs in many variants, all closely related to statistical adaptive clustering procedures. DK uses it to locate clusters in concatenated input-output data. The idea is to locate regions where a local preponderance of sample data indicates a causal relationship. For example, in a simple situation, clusters could arise in a set of data. In a plot of the data, each point on the graph would represent a node that had summarized a large number of data points.

Three clusters are immediately apparent, at least to the eye. It's not always so easy to locate clusters with an algorithm,

***Fred A. Watkins**, Ph.D., president, Hyperlogic (Escondido, CA)*

but algorithms do exist that enjoy a measure of success as cluster-finders. Let's suppose we have found three clusters in the data.

It's easy to see that if the independent variable is near one of the horizontal axis tick marks, the dependent variable value will fall near the corresponding vertical axis tick mark. All we need to formulate rules is some fuzzy set definitions that capture the meaning of the phrase "near a tick mark." DK uses the competitive network clusters to estimate these fuzzy sets by using covariance ellipsoids (a multidimensional version of an ellipse).

Each cluster represents several data samples, and DK uses that data to compute an ellipsoid, which is centered at the centroid of the data set. Each axis of the ellipsoid has a length determined by the covariance matrix derived from the data. This "random ellipsoid" represents both the location and the variability of the data in the cluster: The location estimates the input-output relationship implicit in the data; the variability estimates the accuracy of the estimate.

An ellipsoid determines fuzzy sets by projection onto the coordinate axes. Each input set is an isosceles triangle centered at the projection of the cluster centroid, and the base of the triangle is the projection of the ellipsoid onto that coordinate.

Now we have rules—one derived from each cluster—that connect the input variables to the output variables in the obvious way. If the system has two inputs and one output, the rule corresponding to cluster 7 might read, "If X is x-fuzzy-set-7 and Y is y-fuzzy-set-7 then Z is z-fuzzy-set-7."

Competitive learning is a rapid algorithm in that it tends to reach stochastic equilibrium with exponential speed. It often finds stable clusters in just a few passes through the data, deriving a ruleset in short order.

Optimizing the fuzzy set

The second phase of the DK procedure uses supervised learning to further optimize the fuzzy set definitions obtained in the first step. Supervised learning is the general name of a large class of stochastic algorithms that require both input and desired output data to adapt. The back propagation algorithm is the most popular and well-known example.

Using the fuzzy system responses and the desired output data, an error measure can be calculated. Any consistent error measure is suitable; DK specifically uses mean square deviation. During this phase the supervised algorithm adjusts the centers of the fuzzy sets to minimize the mean square error between the actual and desired system responses.

Using competitive learning, DK rapidly makes a first guess at the ruleset, then refines that guess more slowly with supervised learning. The first pass with competitive learning tends to produce good initial conditions for the supervised learning phase to follow. Good initial conditions tend to decrease the time supervised learning requires to stabilize.

Cluster-finding is a problem when there's a large amount of data. Most existing methods are statistical and require parameter settings or other input from operators.

Improving on the DK method

Leaving the problem of cluster-finding aside, we might ask whether the DK method can be improved, and if so, how it can be accomplished. Since the second phase only tunes the centers of the fuzzy sets, it might be that a larger supervised network that can adjust the shapes as well as locations of the fuzzy sets could further improve system performance. Another possibility is to use annealing or other methods to improve the chance of avoiding local minima during the second phase.

Dimensional complexity is the chief impediment to the successful use of supervised learning (or any other method, for that matter). Suppose the first pass produced five fuzzy sets for each of three input variables. In many cases, output fuzzy sets can be represented by "singletons" and so may not require much adjustment. Then, a 128-cell partition of each input dimension yields a total of 128 x 15 = 1,920 variables to optimize. Even if a 16-cell partition could suffice, there are still 240 variables. Hardware that can handle such networks exists but is very large and expensive.

The second possibility can be implemented on small machines in many ways. Future research will tell us if this kind of approach offers substantial benefits.

An ideal solution must arbitrarily handle many inputs and overcome the problem of local minima. It must do this in a model-free manner— without any prior knowledge of the underlying input-output relationship. Given the current capabilities of mathematical and statistical analysis, it's not likely that this kind of success will occur in the foreseeable future.

output relationships. These data points can be taken from measurements of a running system or written as specifications to be learned. The more data supplied, the more accurately the resulting curve will fit the actual system behavior. You can specify the accuracy of the data supplied, as well as how "fat" the data points are. For example, you can specify a tolerance of 1 percent around any given point as an acceptable range of accuracy for the data. This will effect how long it takes the system to learn the data.

"With NeuFuz, you can specify the accuracy [of the data]," says Emdad Khan, head of intelligent systems for National Semiconductor's Embedded Systems Division. "When the learning is completed, it'll give you a solution that has that much accuracy."

You can then specify a so-called epsilon value that represents the accuracy of the neural net learning. Because of the nature of neural networks, however, this will never be 100 percent accurate. Says Fred Watkins, president of Hyperlogic (Escondido, CA), "Most data is stochastic, so you don't try to reach 100 percent. Instead, you reach for the kind of thing that's in regression, where you want the best line and the best curve through a bunch of data. Specifying an accuracy of 90 percent may yield a completely acceptable result for the learning process while saving a great deal of processing time that might be spent in further reduction."

Data & learning accuracy

There's a difference between the accuracy of the data and the accuracy of the learning: The accuracy of the learning is the allowable difference between the neural net's computed output and the desired output values. This aspect of accuracy depends on the number of data patterns supplied and the number of learning cycles. When all the inputs to the neural net produce a corresponding output within the predefined error margin, the neural net is said to have converged. Convergence can sometimes be plotted as a smooth, descending curve that approaches the epsilon value as learning cycles are repeated. Sometimes, however, the convergence oscillates and the descent curve shows valleys. In actual practice, this curve is mathematically represented as a 3-D surface.

The convergence can reach a point

Toward soft computing

Fuzzy logic and neural networks both work with imprecision. Neural nets find the best curve—if not an exact curve—to fit a set of data. Fuzzy systems let you make generalizations about system behavior in linguistic terms and manipulate them with approximate reasoning. This is what Prof. Lotfi Zadeh, the inventor of fuzzy logic, describes as exploiting the tolerance for imprecision. It's something you either do consciously or unconsciously in many real-world situations.

"The most telling example is high-definition television," says Zadeh. "To squeeze an HDTV signal into a standard TV bandwidth, people exploit the tolerance for imprecise vision. For example, if an object moves, you can't discern the detail. You calculate what the position of various points of that object might be, but you don't transmit that information."

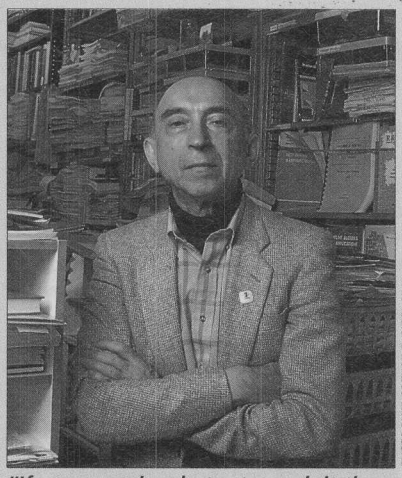


PHOTO BY FLORENCE G. O'DONOVAN

"If you consciously try to exploit the tolerance for imprecision, you can solve many problems that couldn't be solved otherwise," says Prof. Lotfi Zadeh, the inventor of fuzzy logic.

This is an example of exploiting the tolerance for imprecision that doesn't use fuzzy logic or neural nets. But Zadeh is proposing a discipline known as "soft computing" that consciously exploits this tolerance and uses computing approaches specifically tailored to deal with it.

"If you consciously try to exploit the tolerance for imprecision, you can solve many problems that couldn't be solved otherwise," says Zadeh. "This is the guiding principle of what soft computing tries to do."

Soft computing incorporates three main constituents: fuzzy logic, neural networks and probabilistic reasoning. Probabilistic reasoning subsumes such exotic disciplines as belief networks, genetic algorithms, chaos theory and uncertainty management. Fuzzy logic focuses on approximate reasoning while neural networks focus on curve fitting and learning.

The art of soft computing will center on the appropriate application of one or more of these constituents to a given problem. The premise behind soft computing is that traditional hard computing doesn't adequately reflect the imprecision and uncertainty of the real world. It says that trying to achieve precision and certainty in such a world carries an unnecessary cost. On the other hand, tractability and robustness can be achieved at a relatively low cost by acknowledging and exploiting the tolerance for uncertainty and imprecision. The exploration of the possibilities for soft computing in control systems, as well as in such fields as risk assessment, finance and other fields, has just begun. The ultimate results are, at best, uncertain.

where it can't reduce the error with further learning cycles, but still hasn't reached the desired error value. If it gets stuck in this way, it may need more data or the learning algorithm may need to be adjusted. At this point you can either feed it more data and run the algorithm again or implement simulated annealing. Simulated annealing randomizes some variable in the algorithm to jostle the learning process so that subsequent passes can bring the error to a lower value. The amount of error may initially increase before beginning to converge again. Plotting the convergence of such a system over the number of cycles will show a curve that rises and falls as it approaches the specified accuracy.

Learning through repetition

Like NeuFuz, the NeuroFuzzy module from Inform uses a neural network learning technique known as back propagation of error. In addition, NeuroFuzzy supports several other learning methods. The back propagation approach feeds the difference between the net's output and the expected output of the system to adjust the neural net's internal weights as it converges to a solution. Because NeuroFuzzy works with the same data representation as the fuzzyTECH tool, its output is available for editing and user intervention much like a file created with fuzzyTECH. You'll usually start by defining some fuzzy structure that the NeuroFuzzy module can modify, based on the data it's learned.

The fuzzyTECH NeuroFuzzy tool from Inform is an option that can be integrated with the main fuzzyTECH design environment. When you install it, it adds new DLLs to fuzzyTECH so that fuzzyTECH has new menus, new dialog boxes and new functionality within the system. The NeuroFuzzy module is very interactive since it's directly integrated with the fuzzy design tool.

"It's like you have a magic mouse somewhere in fuzzyTECH that modifies parts of the system," says Constantin von Altrock, director of fuzzy logic at Inform. "When you start learning, you can actually see what the module is doing, such as how it modifies and generates membership functions. You can, of course, stop learning and intervene at any time in the process."

The NeuroFuzzy module also has what Inform calls restricted learning. You may already have a very clear idea of how some parts of the system are supposed to work and you don't need to learn data to implement them. So you can select any part of the system to learn. "There may be certain fuzzy logic rules that don't need to be changed," says von Altrock, "or other rules that you don't want the system to touch."

Making fuzzy make sense

Paradoxically, neural networks are rather dumb when it comes to figuring out how to build a fuzzy logic rule base. In National's NeuFuz, for example, you start out by specifying the number of membership functions you want for each input and output. The neural net then learns the data and produces a file that's fed into the rule and membership function generator. This module produces a set of all possible rules. For two inputs with five membership functions each, for example, there would be 25 rules; for three inputs at five membership functions there would be 125 rules, etc.

In fuzzy logic, many rules actually

continued on page 80

1 GIGABYTE MEMORY MODULE

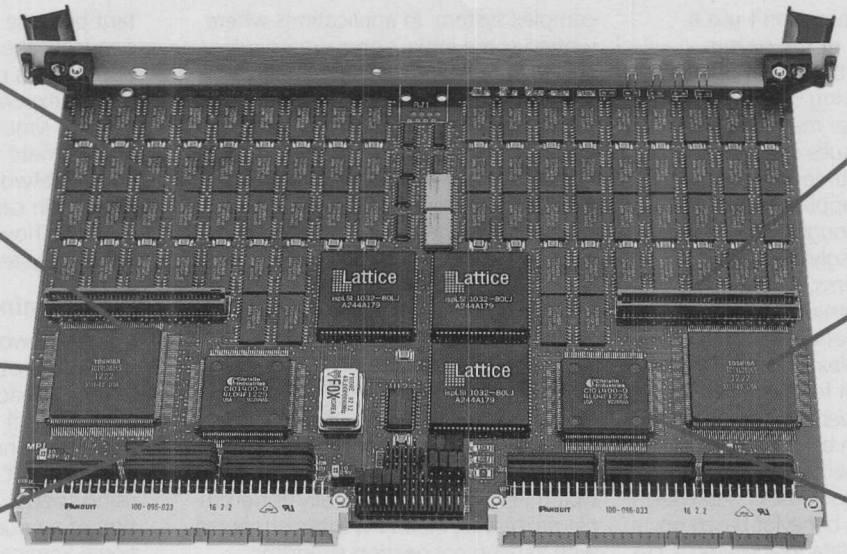
VME/VSB/VME64

1 Gigabyte of FAST DRAM

Chrislin Custom ASIC Chipset

High-Speed VSB Interface

Independent VME/VSB Address Selection



Field Upgradeable to Full 1GB Capacity

High-Speed VME/VME64 Interface

ALU Processing Of Data During Transfers

CI-VME80

- **Up to 80 Mbytes/Sec Transfer Rate on VME64**
- **Up to 40 Mbytes/Sec Transfer Rate on VME/VSB**
- **Fully Supports IEEE 1014D Rev. D & VSB Rev.C**
- **32MB, 64MB, 128MB, 256MB, 512MB, & 1GB**
- **Block Cycle times of 93ns, Single Cycle times of 195ns**
- **Field Upgrade Options, 256MB to 512MB, or 1GB**
- **Enhanced Data Manipulation Functions in ALU Cycles**
- **Access Times of 30ns Write/Read in Block Cycle**
- **Dual-Ported VME/VSB and VME64 in one 6U slot**
- **LIFETIME WARRANTY**

The CI-VME80 is the ultimate memory board for VMEbus systems! Chrislin has designed the CI-VME80 with an onboard ALU to provide enhanced functionality for custom users. Available through AM codes are 64 bit arithmetic and logical functions (ADD,SUB,AND,OR,XOR)that will greatly enhance throughput in many applications. Simply write data to the location to be modified with the proper AM Code and the function is performed during the write cycle.

Chrislin Also Provides The Following VMEbus Memory Boards

CI-VME40

- 4MB, 8MB, 16MB, 32MB & 64MB
- Block Cycle times of 84ns
- Single Cycle times of 195ns
- Block Cycle Access Time of 30ns
- Dual-Ported VME/VSB in one 6U slot

CI-SRAM

- Non-Volatile High Speed VME Memory
- 4MB or 16MB Static RAM
- 16MB EPROM
- Two Selectable Write Protect Banks
- 32 Bit BLT Transfer, 40MB Per Second

CI-VSB-EDC

- Error Detection and Correction
- Single-bit Correction, Double-bit Detection
- Dual-Ported VME/VSB in one 6U slot
- 4MB, 8MB, & 16MB
- VME & VSB configured independently

FOR SPECIFICATIONS, AND ADDITIONAL INFORMATION

Call 1-800-468-0736

Chrislin Industries, Inc.

31312 Via Colinas #108, Westlake Village, CA 91362 USA

TEL: (818) 991-2254 FAX: (818) 991-3490

Providing Top Quality Memory For Two Decades!

Circle 44

Neural networks for fuzzy logic accuracy

Fuzzy logic techniques don't use a mathematical model of the system, which can be difficult, if not impossible, to obtain when working with complex systems. Conventional mathematical model-based techniques can only effectively address linear, time-invariant systems, leaving many applications without viable solutions. Although fuzzy logic has proved successful in solving some of these types of problems, it's generally difficult and time consuming to determine a correct set of rules and membership functions for a complex system. Fine tuning the fuzzy solution for a predictable level of accuracy is even more difficult.

These problems can be effectively eliminated by combining neural networks with fuzzy logic. This approach uses neural networks to learn system behavior based on system input-output data to a desired accuracy, and generates fuzzy rules and membership functions based on the learned system knowledge. The generated fuzzy solution provides the same desired accuracy that the neural network was trained for. Combining these technologies also minimizes system cost by optimizing the number of rules and membership functions.

Language-based reasoning

Fuzzy logic is an approximate reasoning technique that uses human-based language to describe a system's input-output relations, as opposed to crisp reasoning used in conventional artificial intelligence, which requires a "yes" or "no" answer.

Linguistic variables such as "high," "low," "medium," "hot," "cold" and so forth are used to describe the various degrees or levels of the inputs and outputs. System inputs and outputs are related by fuzzy rules using linguistic variables and such rules can describe a system. Experts can use experience to describe the behavior of a system that's being considered using human-based language, which can better express uncertainties, imprecisions and nonlinearities. In this way, fuzzy logic techniques avoid a complex mathematical description of the system, making its implementation simpler and more cost-effective than conventional logic.

The downside to using fuzzy logic in some applications is that it can be difficult to write a correct set of rules and membership functions, particularly for a complex system. In applications where writing some initial approximate rules and membership functions is fairly simple, the challenge is to properly tune these rules and membership functions to get a desired level of outcome accuracy.

The accuracy of the result is checked against the current solution, and initial rules and membership functions undergo iterative tuning to improve it. This doesn't guarantee a solution that meets desired accuracy, however, and it generally takes a long time to develop the solution.

Learning and generalization

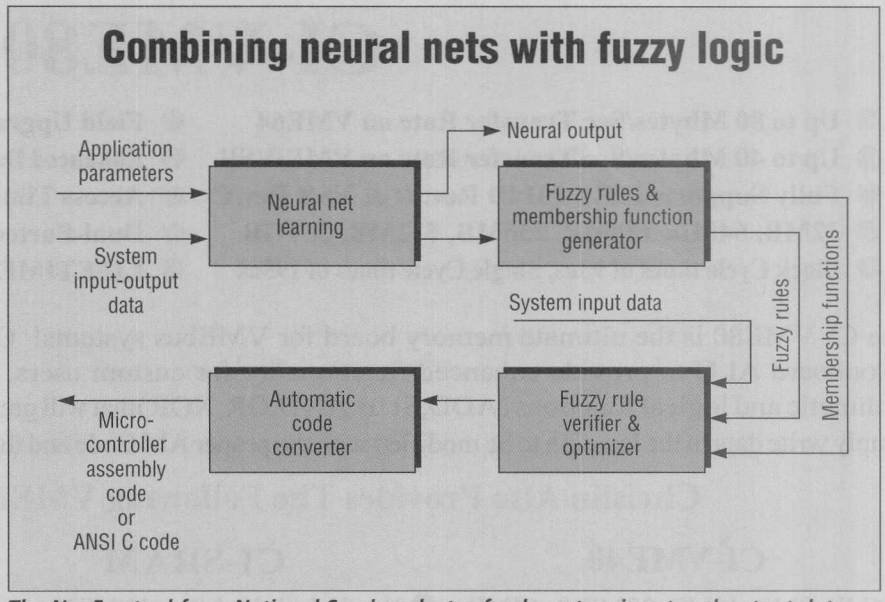
Neural networks can be used to eliminate some of the shortcomings of fuzzy. A neural network is an artificial structure that mimics the human brain using simple processing elements interconnected with varying connection strengths known as weights. A neural network has learning and generalization capabilities, letting it learn a system's input-output behavior. Learning is achieved by varying connection strengths. Once a neural network is trained, it can be used to develop solutions to control the system for any input values within the specified range.

Determining the correct set of input-output data for learning is very important because system performance depends on the learning (training). Input-output data can be obtained by measurements, experience, mathematical models, simulation or from an existing approximate fuzzy solution. Although neural networks have learning and generalization capabilities, it's difficult to understand how weight change really maps into the system input-output relation.

Combining neural with fuzzy

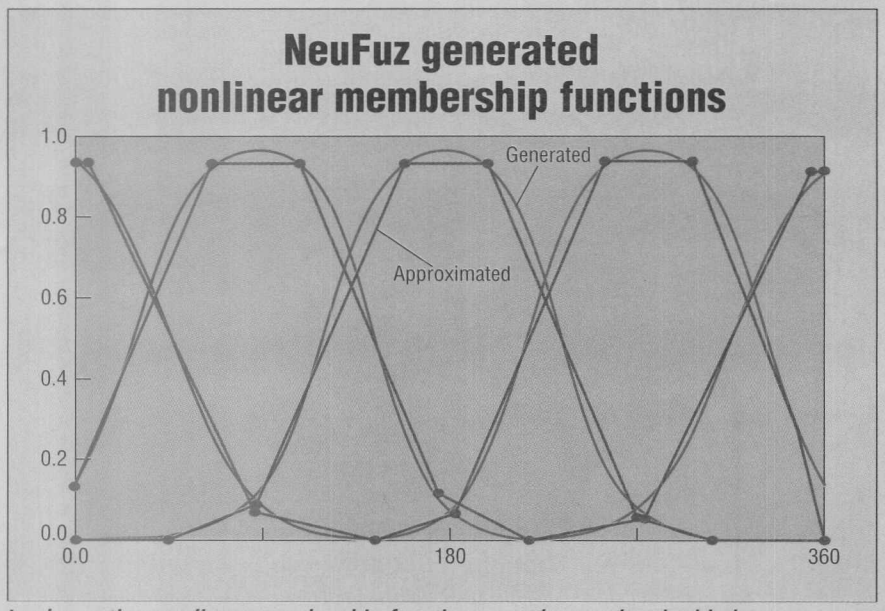
Neural networks and fuzzy logic complement each other by eliminating the limitations of each approach. The combination makes it easy to generate fuzzy logic rules and membership functions. It also makes it easier to fine tune them. By using special architecture and proper design of a neural system, such as in National Semiconductor's NeuFuz tool, learned knowledge can be directly mapped to fuzzy logic. This enables automatic generation of fuzzy logic rules and membership functions.

The derived fuzzy logic solution provides the same accuracy that the neural network was trained for and can easily be implemented on virtually any microcontroller. The combined technology can also optimize the solution (minimizing



The NeuFuz tool from National Semiconductor feeds system input and output data along with application parameters into a neural network. When the data has been learned, the file is translated into a set of all the possible rules. The number of rules depends on the number of inputs and the number of membership functions specified. The rule optimizer and verifier lets the user discard rules that don't significantly contribute to the solution. The system can then generate assembly or C code.

Emdad Khan, *Intelligent Systems Group, Embedded Systems Division, National Semiconductor (Santa Clara, CA)*



Implementing nonlinear membership functions on a low-end embedded processor may be difficult. For such a processor, an approximated (trapezoidal like) membership function would be a better choice.

the number of rules and membership functions) and minimize cost or improve response time. And because of the generalization capability of neural networks, the derived fuzzy solution becomes more robust and reliable. The combined approach can also help minimize the problems associated with neural nets.

NeuFuz is one approach where neural network technology has been effectively combined with fuzzy logic to eliminate some of the problems associated with fuzzy logic. The NeuFuz approach goes a step farther than some other hybrid solutions by using the learning and generalization capabilities of neural networks to achieve higher accuracy and shorter development time. This is accomplished by automatically generating fuzzy rules and membership functions to meet a desired accuracy and optimizing the solution.

Going nonlinear

Fuzzy logic usually uses membership functions that have fixed geometric shapes. Because of the fixed shapes, such membership functions limit system knowledge more in the rule base than in the membership function base. Because of this, as the number of rules increase, so does the system's memory requirements and processing time. Neural networks alleviate this problem by learning nonlinear membership functions. This helps put more system knowledge in the membership function base, reducing the number of rules needed in the rule base and reducing memory requirements.

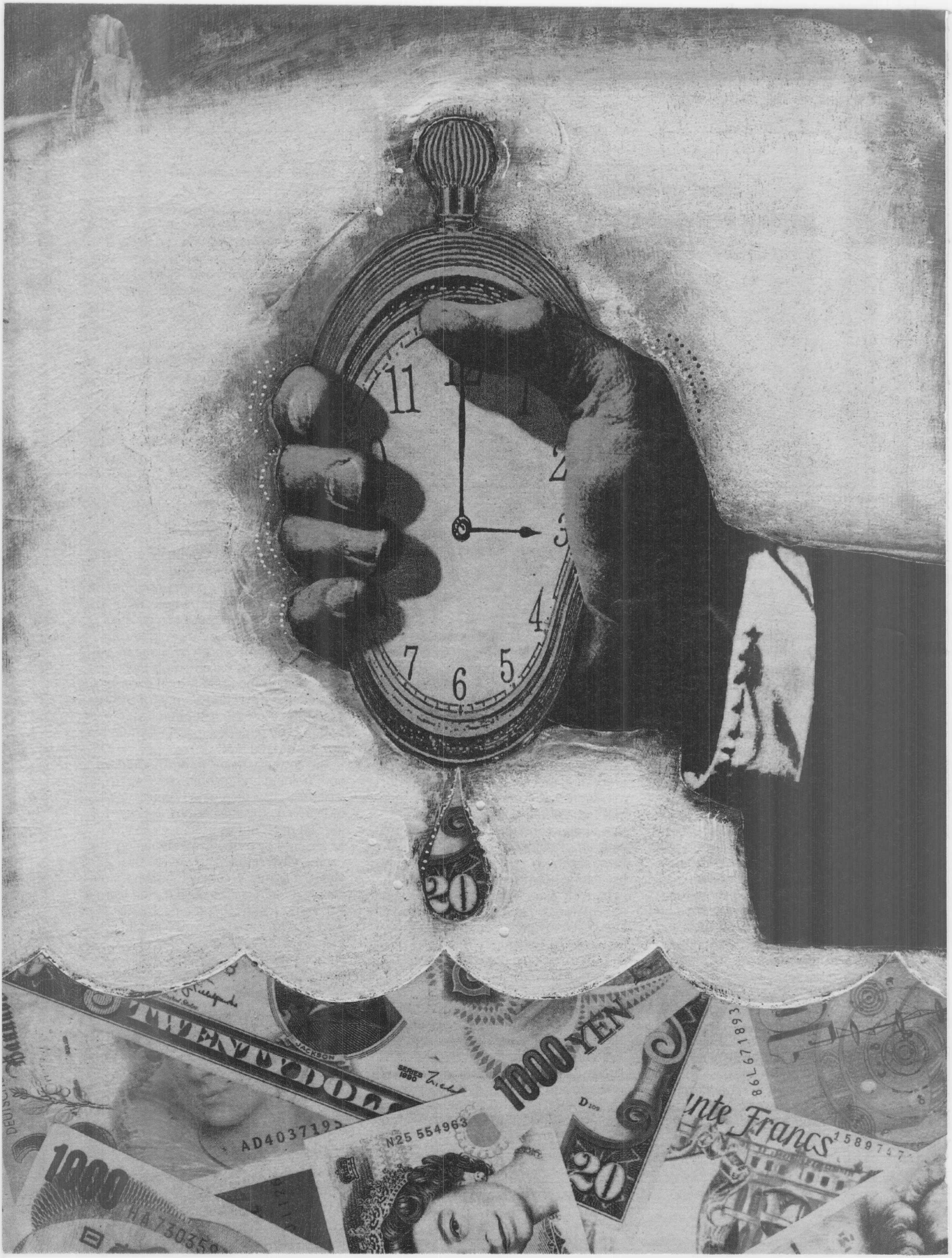
Fuzzy logic uses heuristics in defuzzification, rule inferencing and antecedent processing methods. Although heuristics may work well for many solutions, they don't guarantee satisfactory solutions that operate under all conditions and wide input ranges. A combined neural-fuzzy approach, such as NeuFuz, would use neural network-based nonheuristic fuzzy logic algorithms for defuzzification, rule inferencing and antecedent processing, enabling direct mapping of neural networks to fuzzy logic. This direct mapping to fuzzy logic provides several key advantages:

- The neural network accuracy is fully maintained by the fuzzy logic solution. If the neural network is trained to be 99 percent accurate, the generated fuzzy rules and membership functions will provide the same accuracy if no optimization (i.e., no deletion of rules) is done. The loss of accuracy by optimization can be controlled and made insignificant.
- The fuzzy solution inherits other features of neural networks. Because of the better generalization capability of neural networks, the derived fuzzy solution is more robust and reliable and can work well under a wider range of input values.

A solution's accuracy is usually managed in two steps: during neural network learning and during optimization. The neural network can be trained to a desired accuracy by specifying the accuracy before learning begins. Learning continues until the desired accuracy is obtained. For some types of data, the learning may not be completed at first trial. In such cases, some key parameters—such as learning rate and initial weights—can be adjusted to help converge the neural network.

Optimization steps can begin once the learning process is completed to the desired accuracy. At this point, insignificant rules can automatically be detected and deleted, which may cause some loss of accuracy. But if this loss is insignificant, further optimization can be made by deleting more rules, which simply involves manipulating the optimization parameters. On the other hand, some of the deleted rules after the first optimization step may be re-added if accuracy falls below the desired level.

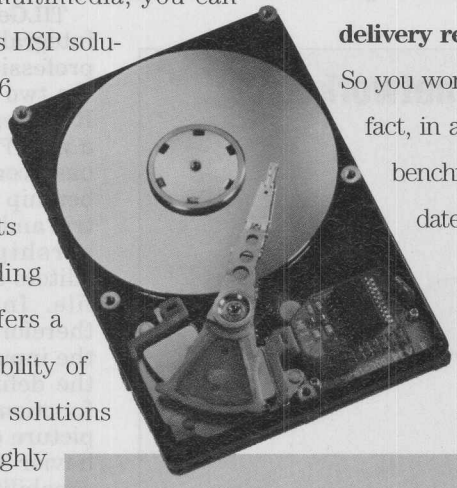
The accuracy needs to be checked for both training and test data sets. Accuracy for the training data is guaranteed by learning. This may not be the case for the test data, but its accuracy can easily be ensured by properly manipulating the neural network learning phase. This means a tighter accuracy than what's ultimately desired can be used during learning. It also means that the neural network can be retrained with a modified training set that includes some of the previous test data that had a less-than-desired accuracy. In any case, a desired accuracy of the final solution can easily be obtained.


11
2
3
4
7
6
5
20
1000 YEN
20
Francs

Squeeze the most out of your design cycle.

You know time is money. And at Texas Instruments, we can help you speed your product to market. Meaning you are free to do what you do best – create the next breakthrough products.

Be first to market with leading-edge designs. From notebooks and PDAs to wireless communications and multimedia, you can harness the power of TI's DSP solutions, low-power 486 microprocessors and mixed-signal devices to design unique products fast. Our industry-leading TMS320 DSP family offers a road map with the flexibility of software-programmable solutions that can migrate into highly integrated, custom DSPs that deliver cost-effective high performance. And compatible logic families and memory devices also help you move your designs easily from 5 V to 3 V.



Total Integration in action.

Seagate Technology put the power of Total Integration to work when creating their hard disk drives. In addition to combining TI's high-performance DSP cores and application-specific logic with memory and analog cells, TI had the tools and service to help Seagate produce a highly differentiated product. And TI's on-time delivery made a difference. TI provides high-volume, quick-ramp manufacturing capability to meet Seagate's needs as an industry leader.

Technical support that gets you up to speed, fast. Move quickly from selection to prototype with clear, concise technical information and real-world application notes. And C compilers, emulators, development tools, behavioral models and SPICE rules can help you create DSP solutions in record time.

Setting the benchmark: A 94% on-time delivery record. At TI, we deliver what we promise. So you won't be left waiting for your ICs to arrive. In fact, in an expanding market, we set an industry benchmark by having a 94% on-time-to-commit-date delivery record.

Win the race to market. Getting to market faster with a better product. That's what business is all about. And with TI, you'll find a unique blend of silicon, support and service. We call it Total Integration.™ You'll call it your competitive edge. To find out more and receive a free subscription to *Integration*, simply call **1-800-477-8924, ext. 3922.**

EXTENDING YOUR REACH™

TEXAS INSTRUMENTS

continued from page 74

make small contributions to the output. To build an efficient and robust system, you must figure out which rules to discard. This is often done the other way around by the designer who states how the system will work in natural language and from the designer's own knowledge. When the neural net presents you with all the possible rules, you're faced with the problem of figuring out which ones to get rid of.

Four inputs, one output

NeuFuz currently supports four inputs and one output which, at five membership functions each, could generate 3,125 rules. Most of these rules, however, wouldn't be used. The optimizer and verifier module supports a deletion factor that automatically discards rules that fall below a defined threshold. The deletion factor is a value between zero and one that indicates how much a given rule contributes to a solution. Setting a deletion factor of 0.1, for example, would eliminate rules that contributed no more than 1 percent to the solution.

The verifier/optimizer also lets you edit the membership functions and test the selected rule base against a recall file. The recall file is a set of input data that tests to see if the design results correspond to the expected behavior.

The files produced by Inform's NeuroFuzzy module are descriptions of a fuzzy logic system that can be accessed by all other parts of the design system. These include the linguistic variable editor, the rules editor, the graphic analyzer and the interactive debugger.

The rules editor displays a graphic representation of the rules matrix known as a fuzzy associative memory (FAM), which also represents the weights of the rules by display color and numerical value. By examining the rules matrix you can select sets of rules and evaluate their performance using the simulator or the graphic analyzer. You can also modify membership functions and put together a final set of rules that satisfy the performance requirements.

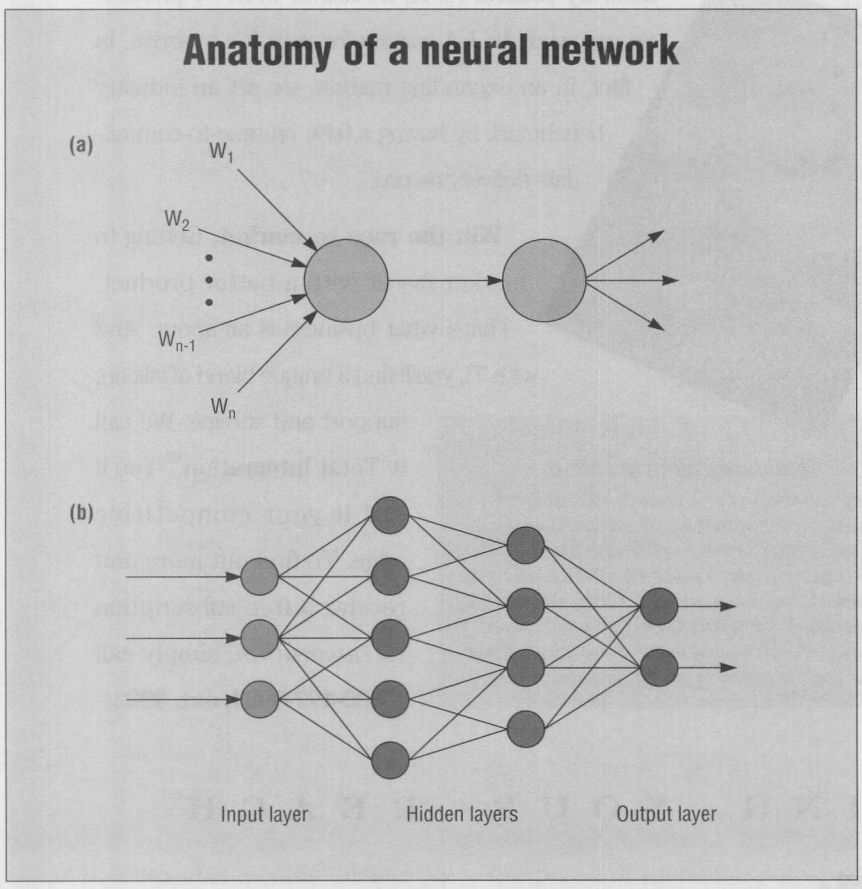
Togai InfraLogic's TILGen tool uses neural network learning to generate rules from fuzzy input objects that have already been defined. Using Togai's TILShell 3.0 development environment, you can supply a file written in Togai's Fuzzy Programming Language (FPL). The FPL file contains input and output objects that are the system variables with sets of membership functions but no rules. You then supply a separate file containing input and output vector data. The data can be created by measuring the actual behavior of a system or can be specified as the system's desired behavior.

TILGen, which is supplied as an integral part of Togai's TILShell 3.0 professional edition, then applies the two files to the neural network. It processes the files and generates a new FPL file that contains a rule base consisting of rules and membership functions. TILShell rule matrix and table editors, and the membership function and variable editors can access the resulting FPL file. In the Togai environment, therefore, you need to be clear about the input and output variables, and the definition of their membership functions. You also must have a clear picture of the system's expected behavior to use the neural network capability of TILGen.

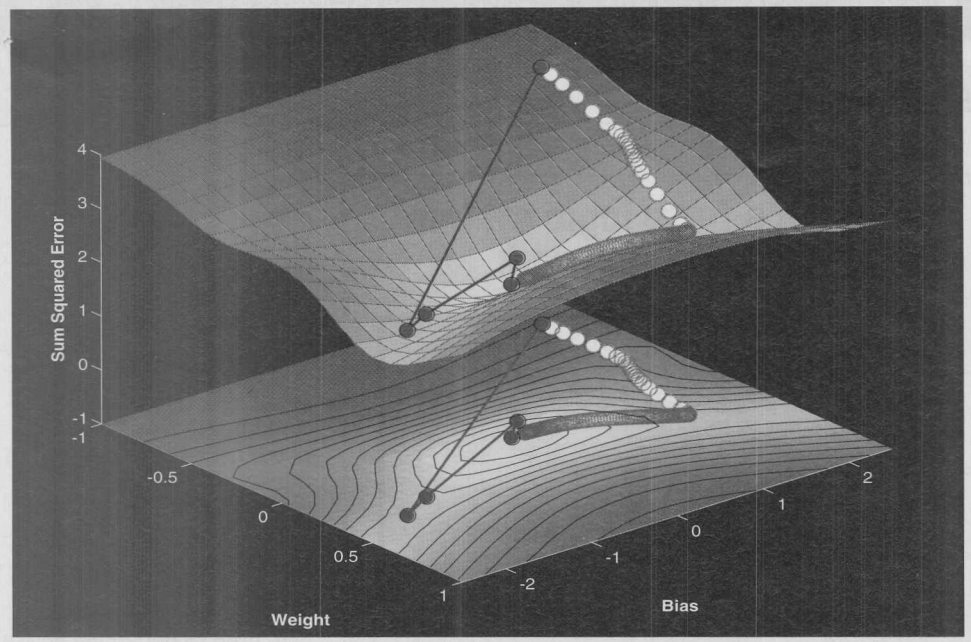
Detailed specs required

All the tools that use neural net technology require detailed specifications of system behavior in terms of input and output data. If that data is taken by measuring some already-running system controlled by some other method, the tools don't necessarily require that you understand the system's behavior. They can actually help provide a better understanding of that behavior.

The same is true of classical control approaches that use a mathematical model to develop a control strategy. Either one must describe the system's behavior with a set of detailed equations or derive those equations from data. Tools that analyze system performance data for classical control strategies include



A neural network consists of layers of individual neurons. A neuron (a) takes input from many other neurons and sends output to many others (b). Each input is multiplied by its weight. When the sum of the weighted inputs (W_1, W_2, . . . W_n) reaches a threshold value, the neuron "fires." As the neural net reads the training data set, it adjusts its internal weights so that the inputs produce an output that's close to the desired value. Each pass through the entire training set is a learning cycle.



MATLAB graphics enhance understanding of neural network behavior. This plot compares training rates for standard backpropagation (white, 108 steps) and the fast Levenberg-Marquardt algorithm (blue, 5 steps). Each trace illustrates the number of steps from initial conditions to the minimum error.

MATLAB
Picture the Power

Neural Network Toolbox

NEW 2.0 FEATURES

- *Levenberg-Marquardt fast backprop training*
- *Radial basis networks*
- *Adaptive linear networks*
- *Elman recurrent networks*
- *Learning vector quantization*

PROVEN FUNCTIONALITY

- *Backpropagation, perceptron, linear, recurrent, associative, and self-organizing networks*
- *Supervised and unsupervised learning rules*
- *Competitive, limit, linear, sigmoid, and user-defined transfer functions*
- *Performance analyses & graphs*
- *Unlimited layers, elements, & interconnections*
- *Fully open and customizable*
- *Runs on PCs, Macs, and workstations*

Introducing Version 2.0 of the Neural Network Toolbox

Both newcomers and experts have found the MATLAB® Neural Network Toolbox to be ideal for exploring and applying neural networks to fields as diverse as signal processing, nonlinear control, and financial modeling.

Now, version 2.0 offers five new network paradigms, major speed improvements, and expanded support for experimentation.

Beyond backpropagation

The MATLAB Neural Network Toolbox offers over fifteen proven network architectures and learning rules. You can choose among feed-forward, recurrent, associative, and self-organizing network paradigms, using supervised or unsupervised training.

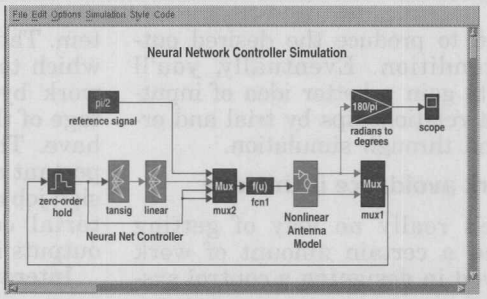
The first neural network engineering environment

The Neural Network Toolbox is a complete engineering environment for neural network research, design, and simulation. Unlike other neural network packages, you never work in isolation from real world considerations.

Flexible data import and transformation functions simplify preprocessing of input data. Powerful analysis and advanced graphics help you evaluate network behavior and performance in the context of complete system designs.

With no limits on network size or connectivity, you can implement networks as complex as your application demands.

You can easily change any architecture, learning rule, or transfer function — or add new ones — without writing a single line of C or Fortran.



A SIMULINK® nonlinear simulation of a control system designed with the Neural Network Toolbox. You can also use neural networks with other MATLAB toolboxes such as Image Processing to create optimum solutions.

Not just for experts

You don't have to be an expert to use the Neural Network Toolbox. The User's Guide makes learning a breeze, with tutorials and examples for each type of network.

The Ultimate Technical Computing Environment™

MATLAB is an extensible, interactive Technical Computing Environment that seamlessly integrates computation, visualization, and modeling — the ideal way to discover the power of neural networks.

For a technical data sheet, call **508-653-1415**

The MATH WORKS Inc.
24 Prime Park Way
Natick, MA 01760-1500
Fax: 508/653-6284
Email: info@mathworks.com
Mosaic: http://www.mathworks.com

NeuroFuzzy technologies

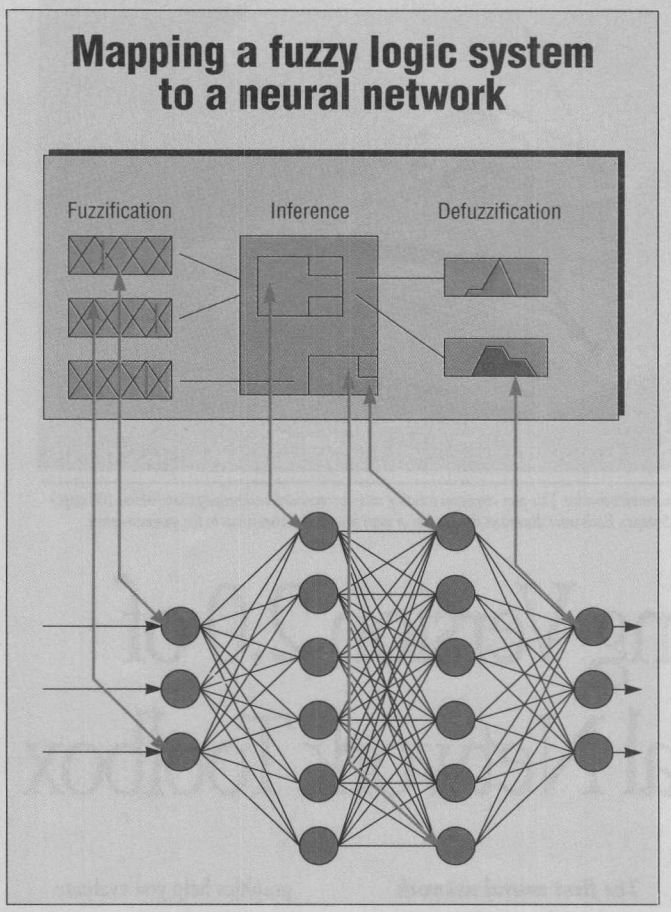
The key benefit of fuzzy logic is that it lets you describe system behavior with simple "if-then" relations. In many applications, this gets you a simpler solution in less time. You can also use all available engineering know-how to directly optimize the performance. While this is certainly the beauty of fuzzy logic, it's also a major limitation. In many applications, the knowledge that describes a system's behavior is contained in data sets. Here, the designer has to manually derive the "if-then" rules from the data sets, which imposes a major effort with large data sets.

Many people have suggested neural nets as a means of training a desired system behavior from data sets. But neural net solutions have produced better results than other methods in only a few applications.

In the dark

Neural nets are rarely used in applications for several reasons. First, a neural net solution remains a "black box." You can't interpret what causes a certain behavior, or manually modify a neural net to change a certain behavior. Second, neural nets require prohibitive computational effort for most mass-market products. Third, selecting an appropriate net model and setting the parameters of the learning algorithm is still a "black art" and requires a lot of experience. But of all these, the lack of an easy way to verify and optimize a neural net solution is probably the major limitation.

Both neural nets and fuzzy logic are powerful design techniques with their own strengths and weaknesses. Neural nets can learn from data sets while fuzzy logic solutions are easy to verify and optimize. If you compare these properties, it's apparent that a clever combination of the two technologies can deliver the best of both worlds. Combine the explicit knowledge representation of fuzzy logic with the learning power of neural nets, and you get NeuroFuzzy.



The NeuroFuzzy Module maps a neural net to a fuzzy logic system. This lets the powerful neural net learning algorithms be used with fuzzy logic systems.

Ten years ago, there was a major milestone in the development of neural net technology: the creation of the so-called error back propagation algorithm. This learning approach proved to be very powerful and most neural net applications use it today. First, it computes the output values of the neural net for the input values of the current training example. Then, it compares these output values to the desired output value given by the training example. The difference, known as error, is now used to determine how a neuron in the net is modified. The mathematical map of the error back into the net is known as error back propagation.

Since this learning approach has already proved that it's usable, why not use it for fuzzy logic systems as well? This isn't as easy as it seems. The problem is that to determine which neuron had what effect, the error back propagation algorithm has to mathematically differentiate the transfer function of the neurons. If you want to apply the error back propagation algorithm to a fuzzy logic system, the problem is that the fuzzy logic inference generally isn't differentiable.

To overcome this problem, some neuro-fuzzy development tools use an extended fuzzy logic inference method based on Fuzzy Associative Memories (FAMs). These FAMs can be considered a generalization of normal fuzzy logic rules. Because they can also be mathematically transferred to neurons in a neural net, a modified error back propaga-

***Constantin von Altrock**, director, fuzzy logic, Inform Software (Aachen, Germany)*

the MATRIXx tools from Integrated Systems Inc. (ISI—Santa Clara, CA) and Matlab from The Math Works (Natick, MA).

If you start designing a system using fuzzy logic alone, you might not start with a clear idea of just how the system will behave for all input conditions. Or you might not know just which combination of inputs will be needed to produce the desired output condition. Eventually, you'll have to gain a better idea of input-output relationships by trial and error and through simulation.

Work avoidance is the idea

There's really no way of getting around a certain amount of work involved in designing a control system. The choice is more a matter of which tools to apply to avoid extra work by taking maximum advantage of the information you already have. This becomes especially important as the complexity of the design job and the attendant combinatorial complexity of inputs and outputs increases.

Interestingly, National Semicon-

tion algorithm can be used.

When you start with a NeuroFuzzy design, the first step is to obtain the data sets that represent the desired system behavior. Each data set gives sample output values for a given combination of input variables. The NeuroFuzzy training process starts with a fuzzy logic system. If you haven't set up your fuzzy logic system yet, the NeuroFuzzy Module can automatically set up an initial system for you. It analyzes the data sets and proposes a system structure. You can either accept this structure or modify it, before fuzzyTECH generates this system with default membership function definitions and default rules.

In the next step, you select the parts of the system that the NeuroFuzzy Module may modify. This is a major advantage over neural learning, since it lets you control the training process. This is especially true when you use the NeuroFuzzy Module to optimize an existing fuzzy logic system—you can exclude parts of the system from learning. Also, if you have data sets representing different aspects of the system performance, you may direct the NeuroFuzzy Module to learn different parts of the system from different data sets.

When you start training, graphical editors show how the NeuroFuzzy Module modifies rules and membership functions. You can interrupt the training at any time, either for system analysis or for manual modification. You can continue the training with the current or a different data set at any time. Also, you can set termination conditions if you don't want to manually control the training progress. The training result is a fuzzy logic system.

Another advantage of NeuroFuzzy is that the generated code is much more efficient compared to neural nets. Computing a NeuroFuzzy system on a microcontroller or a PC may require as little as 0.1 ms and 1 kbyte of memory. In contrast, the neural net implementation of a similar complex system may use about three times as much code and 100 times as much computing time. This lets you integrate NeuroFuzzy solutions in most real-time applications.

In addition to automatic generation of fuzzy logic systems using sample data, NeuroFuzzy lets you tune systems while they're running, either to increase their performance or to cope with changing parameters in the environment. If you use fuzzy logic, you define the system behavior explicitly; with NeuroFuzzy you define it implicitly by examples. With NeuroFuzzy adaptive systems, you only tell it what the goal of its behavior is.

A case study

AEG of Germany has used fuzzyTECH's NeuroFuzzy Module to reduce the water and energy consumption of its new generation of washing machines. Using the signal of an existing sensor in the washer's drum, a NeuroFuzzy system estimates the amount and the type of laundry.

If washing experts know the amounts and types of laundry, they can determine how much water is needed in the washing and rinsing steps. But the washing machine doesn't know the type and amount of laundry, and having the user enter this information makes the machine too complicated. And adding special sensors would be too expensive. As a result, the machine is programmed for the maximum load possible, wasting water and energy.

AEG turned to Inform Software for a different solution. After the machine is loaded with the laundry, the water valve opens and fills the drum to a defined level. The drum is then rotated in a certain rhythm, dipping the laundry in the water. The laundry sucks up water, with the speed and volume depending on the type and amount of laundry. This is measured by an analog pressure sensor that previously was only used for water level control.

The resulting sensor signal is difficult to interpret. While the AEG washing experts know the amount of water required for different laundry loads, they have no experience in interpreting the sensor signal curves. On the other hand, many washing experiments have been conducted with known laundry loads and recorded sensor signal curves.

This is where the fuzzyTECH NeuroFuzzy Module solved the problem. We gave the washing experts data sets of the experiments showing only the laundry load. The washing experts recommended how much water should be used in the washing steps for the given load.

These data sets were used to train a fuzzy logic system with the NeuroFuzzy Module. The fuzzy logic system's input is the sensor signal curve recorded in the washing experiments. The desired outputs for the fuzzy logic system are the water amounts that will be used in the washing steps. The laundry load wasn't used for training.

By analyzing the input data patterns, initial fuzzy logic membership functions are generated. The actual learning process generated 157 rules for the fuzzy logic system. Preparing the data sets, setting up the system, NeuroFuzzy training and final test only required two man weeks of effort.

Field tests prove that this solution saves 20 percent of the average water consumption in everyday use compared to AEG's previous machine, which already was an industry leader in water conservation. Since all European washing machines use electricity for water heating, average energy consumption was cut by 20 percent as well. The NeuroFuzzy solution uses the existing sensor and runs on the existing 8-bit microcontrollers. The product is a major market success for AEG.

ductor found that its NeuFuz tool helped some customers achieve their design goals and simplify the hardware design by reducing the number of sensors. The present NeuFuz tool provides four inputs and one output. On the surface, that seems rather limited and the company is considering a six-input/two-output version. "One reason we use a simple approach is because it satisfies 90 percent of the applications," says National's Khan.

According to Khan, customers using NeuFuz have started off with six, seven or eight inputs. "But when they started measuring data on those inputs, they found that some were so closely interrelated—the correlation was so high—that having those separate inputs wasn't necessary." It was often possible to average such interrelated inputs and get to an acceptable performance while reducing the number of sensors providing input. "One objective is to reduce the amount of sensing you have to do and reduce cost," says Khan.

Using neural nets with fuzzy logic

systems certainly has potential benefits. One major question, however, is how to make those benefits available to users without requiring the specialized skills involved with neural network technology. "My experience with neural nets," says Hyperlogic's Fred Watkins, "is that they are basically wild horses and one must be a skilled rider in order to tame them." In addition to neural nets, "any statistical procedure that can adjust the parameters successfully is a viable candidate" for tuning and/or generating fuzzy systems, he notes.

Hyperlogic's CubiCalc tool combines a fuzzy development and decision support system with a simulation engine. Watkins says a lot of people have expressed interest in a fuzzy/neural hybrid but some method other than a neural net might be a more appropriate method. "I think I know how it can be done without a lot of neural net mumbo-jumbo," he says. "To the degree that it involves neural nets, the net will be under the hood." The usability of any such tools will depend on the extent they can aid the user in finding correlations within sets of raw data and then using that data to construct effective control strategies. At the moment, neural nets appear to have a leg up as compliments to fuzzy tools, but time and the verdict of the users will ultimately decide. ■

For more information about the technologies, products or companies mentioned in this article, call or circle the appropriate number on the Reader Inquiry Card.

Hyperlogic
Escondido, CA
(619) 746-2765 ..... **Circle 207**

Inform Software
Aachen, Germany
(708) 866-1838 ..... **Circle 208**

Integrated Systems Inc.
Santa Clara, CA
(408) 980-1500 ..... **Circle 209**

The Math Works
Natick, MA
(508) 653-1415 ..... **Circle 210**

National Semiconductor
Santa Clara, CA
(408) 721-5000 Ext. 7136 ..... **Circle 211**

Togai InfraLogic
Irvine, CA
(714) 588-3800 ..... **Circle 212**